Garbage Collector

- Klasse vs. Objekt
- Exemplar einer Klasse mit dem new-Operator
- Garbage-Collector (GC)
- Arbeitsweise des Garbage-Collectors (GC)

Klasse vs. Objekt

- Klasse: beschreibt Aussehen eines Objekts
- Objekte entsprechen Elementen und Klassen den Mengen, in denen die Objekte enthalten sind
- Diese Objekte haben Eigenschaften
- Es gibt Möglichkeiten, diese Zustände zu erfragen und zu ändern

Exemplar einer Klasse mit dem new-Operator

- neue Objekte in Java: ausdrücklich mit new-Operator

```
new java.awt.Point();
```

- new-Operator wird gefolgt von Name der Klasse
- Klassenname (hier voll qualifiziert) - Point befindet sich im Paket java.awt
  - Paket: Gruppe zusammengehöriger Klassen Schreibweise kann durch imports gekürzt werden Hinter dem Klassennamen sind runde Klammern für den Konstruktoraufruf

Exemplar einer Klasse mit dem new-Operator

- **Konstruktoraufruf**: eine Art Methodenaufruf
  - Es werden Werte für die Initialisierung des frischen Objekts übergeben
- Speicherverwaltung von Java reserviert für neues Objekt freien Speicher
- Gültiger Aufruf des Konstruktors:
  - new-Ausdruck gibt Referenz auf Objekt zurück

Exemplar einer Klasse mit dem new-Operator

- **new** Operator:
  - Laufzeitsystem reserviert es so viel Speicher, dass alle Objekteigenschaften und Verwaltungsinformationen Platz finden
  - Speicherplatz nimmt Laufzeitumgebung vom Heap.
  - (vordefinierte Maximalgröße: nicht beliebig viel Speicher vom Betriebssystem abgreifbar)

Exemplar einer Klasse mit dem new-Operator

- Alternativen zum **new** Operator:
  - newInstance()
  - clone() neues Objekt als Kopie eines anderen Objekts erzeugen
  - String-Konkatenation mit + : Compiler setzt new einsetzen um neues String-Objekt anzulegen

Exemplar einer Klasse mit dem new-Operator

- System nicht in der Lage, genügend Speicher für ein neues Objekt bereitzustellen:
  - **Garbage-Collector** versucht in letzter Rettungsaktion, alles Ungebrauchte wegzuräumen
  - Immer noch nicht ausreichend Speicher frei: Laufzeitumgebung generiert OutOfMemoryError und bricht Abarbeitung ab

Exemplar einer Klasse mit dem new-Operator

- Heap- Speicher
  - Hier werden Objekt Variablen und Instanzen gespeichert

Exemplar einer Klasse mit dem new-Operator

- Stack – Speicher (Stapelspeicher)
  - Lokale Variablen
  - Methodenaufruf mit Variablen: Argumente kommen vorm Methodenaufruf auf den Stapel
  - Methode kann über Stack auf die Werte lesend oder schreibend zugreifen.
  - Endlose rekursive Methodenaufrufe: ist maximale Stack-Größe erreicht kommt es zu Exception vom Typ java.lang.StackOverflowError
  - Mit jedem Thread ein JVM-Stack assoziiert: bedeutet das das Ende des Threads

Garbage-Collector (GC)

- GC ist ein nebenläufiger Thread
- GC testet ob Objekte auf dem Heap noch benötigt
- Objekte nicht benötigt ➔ sie werden gelöscht
- Objekt nicht referenziert (benötigt): Garbage-Collector (GC) gibt reservierten Speicher frei
- java-Schalter –verbose gc:

  Konsolenausgaben, wenn GC nicht referenzierte Objekte erkennt und wegräumt
- Wegnahme der letzten Referenz: Tod des Objekts

Garbage-Collector (GC)

- Einsatz eines GC verhindert zwei Probleme:
  1. Ein Objekt kann gelöscht werden, aber die Referenz existiert noch (engl. dangling pointer)
  2. Kein Zeiger verweist auf ein bestimmtes Objekt, dieses existiert aber noch im Speicher (engl. memory leak)
- Dekonstruktoren kennt Java nicht
- finalize()-Methode: Ähnlich zum Dekonstruktor

Arbeitsweise des Garbage-Collectors (GC)

- GC ist unabhängiger Thread mit niedriger Priorität
- GC verwaltet Wurzelobjekte, von denen aus das gesamte Geflecht der lebendigen Objekte (der sogenannte Objektgraph) erreicht werden kann
  - Wurzel des Thread-Gruppen-Baums
  - lokalen Variablen aller aktiven Methodenaufrufe (Stack aller Threads).
- GC markiert und entfernt nicht benötigte Objekte

Arbeitsweise des Garbage-Collectors (GC)

- HotSpot-Technologie (Objekte Anlegen - sehr schnell)
- HotSpot verwendet generationenorientierten GC
  - Zwei Gruppen von Objekten mit unterschiedlicher Lebensdauer (meisten Objekte sterben sehr jung, die wenigen werden sehr alt)
- Strategie:
  - Objekte werden im »Kindergarten« erzeugt, dieser wird oft nach toten Objekten durchsucht und in der Größe beschränkt

Arbeitsweise des Garbage-Collectors (GC)

  - Überlebende Objekte kommen nach einiger Zeit aus dem Kindergarten in eine andere Generation
  - Diese wird vom GC selten durchsucht
- GC arbeitet ununterbrochen und räumt auf.
- Beginnt nicht erst mit der Arbeit, wenn es zu spät und der Speicher schon voll ist

Arbeitsweise des Garbage-Collectors (GC)

- Szenario: GC wird nicht mehr benötigtes Objekt hinter der Referenzvariablen ref entfernen, wenn die Laufzeitumgebung den inneren Block verlässt

```
{
  StringBuffer ref = new StringBuffer();
}
// StringBuffer ref ist frei für den GC
```


Arbeitsweise des Garbage-Collectors (GC)

- In fremden Programmen sind mitunter Anweisungen wie die folgende zu lesen: `ref = null;`
- Oft unnötig: GC weiß, wann der letzte Verweis vom Objekt genommen wurde.
- Anders, wenn Lebensdauer der Variablen größer (Objekt- oder statischen Variablen) oder wenn diese in einem Feld referenziert

Arbeitsweise des Garbage-Collectors (GC)

- Wird referenziertes Objekt nicht mehr benötigt:
  - Variable (oder der Feldeintrag) sollte mit null belegt werden
  - GC würde andernfalls das Objekt aufgrund der starken Referenzierung nicht wegräumen
  - GC findet jedes nicht referenziertes Objekt
  - Fähigkeit zur Divination (Wahrsagen), Speicherlecks durch unbenutzte, aber referenzierte Objekte aufzuspüren hat GC nicht

Referenz mit null belegen

```
1. public class GarbageTruck {
2.   public static void main(String [] args) {
3.     StringBuffer sb = new StringBuffer("hello");
4.     System.out.println(sb);
5.     // The StringBuffer object is not eligible for collection
6.     sb = null;
7.     // Now the StringBuffer object is eligible for collection
8.   }
9. }
```

Lebenszeit von Objekten (Methoden)

```
import java.util.Date;
public class GarbageFactory {
   public static void main(String [] args) {
      Date d = getDate();
      doComplicatedStuff();
      System.out.println("d = " + d);
   }

   public static Date getDate() {
      Date d2 = new Date();
      StringBuffer now = new StringBuffer(d2.toString());
      System.out.println(now);
      return d2;
   }
```


Objektinsel

```
public class Island {
  Island i;
  public static void main(String [] args) {

    Island i2 = new Island();
    Island i3 = new Island();
    Island i4 = new Island();

    i2.i = i3;   // i2 refers to i3
    i3.i = i4;   // i3 refers to i4
    i4.i = i2;   // i4 refers to i2

    i2 = null;
    i3 = null;
    i4 = null;

    // do complicated, memory intensive stuff
  }
```

GC explizit aufrufen

```
1.  import java.util.Date;
2.  public class CheckGC {
3.     public static void main(String [] args) {
4.        Runtime rt = Runtime.getRuntime();
5.        System.out.println("Total JVM memory: "
                              + rt.totalMemory());
6.        System.out.println("Before Memory = "
                             + rt.freeMemory());
7.        Date d = null;
8.        for(int i = 0;i<10000;i++) {
9.           d = new Date();
10.          d = null;
11.       }
```


GC explizit aufrufen

```
12.       System.out.println("After Memory = "
                             + rt.freeMemory());
13.       rt.gc();   // an alternate to System.gc()
14.       System.out.println("After GC Memory = "
                             + rt.freeMemory());
15.    }
16. }
```

```
Total JVM memory: 1048568
Before Memory = 703008
After Memory = 458048
After GC Memory = 818272
```

Methode finalize()

- Methode finalize() ist in der Klasse Object definiert
- Folglich besitzt jedes Objekt eine geerbte Methode finalize()
- Methode finalize() der Klasse Object hat allerdings einen leeren Methodenrumpf. Der Garbage Collector ruft die Methode finalize() auf, bevor er ein Objekt aus dem Speicher entfernt.

Methode finalize()

- Durch Überschreiben der Methode finalize() könnten beispielsweise nicht mehr benötigte Ressourcen für ein Objekt freigegeben werden.
- Da weder definiert, wann der Garbage Collector Objekte aus dem Speicher entfernt, noch sichergestellt ist, dass der Garbage Collector auf jeden Fall eine Speicherbereinigung vor der Beendigung eines Programms durchführt, ist generell von der Verwendung der Methode finalize() abzuraten.